

A METHOD AND APPARATUS FOR AUTOMATIC UPDATING OF PRINTER CONFIGURATION AND STATUS DATA

BACKGROUND OF THE INVENTION

Print drivers have long been used to format and transfer computer data from a source of data such as a computer to an output device such as a printer. However, because printers from different manufactures typically have different software and hardware configurations, print drivers typically need to be customized to match the particular printer configuration. Users enter relevant data into the computer to customize the print drivers.

User entry of printer customization data can be time consuming, and is prone to user error. Thus bi-directional drivers that are capable of receiving and processing printer status and configuration data have been implemented.

However, the most widely used operating system, the Windows operating system from Microsoft Corporation of Redmond, Washington, are not designed to support bi-directional drivers. Current implementations of bi-directional drivers in Microsoft OS systems circumvent the spooler to enable data flow in two directions. Circumventing the spooler does not conform to Microsoft specifications. Failure to conform to Microsoft specifications creates problems regardless of the Windows operating system used. In Windows NT systems where many clients communicate to printers through a print server, the avoidance of the print spooler prevents the updating of printer information in the print server because typically, only a system administrator may change the printer configuration settings in the printer server. In Windows 95 and Windows 98 systems that utilize a print server, the print server data is not updated because Windows assumes that the print configuration data is local client data and thus fails to update the print server.

Thus an improved bi-directional print driver is needed.

05631869 "00000000

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a networked system including a client device that communicates with a printer through a print server.

Figure 2 is a flowchart that illustrates operation of a bi-directional print driver in a data "push" configuration.

Figure 3 shows a schematic view of possible data flows in a print provider.

Figure 4 shows a schematic view of a possible port monitor architecture including possible data flows for use in a Windows operating system.

Figure 5 shows one example of a possible user interface displays for setting up port information and bi-directional communication settings.

Figure 6 shows a second example of a possible user interface display for adjusting polling information used in monitoring printer status and configuration data.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 illustrates a networked system including a printer 104 that is coupled to a print server 108. Print server 108 transfers data to be printed from multiple clients 108, 112, 116 to printer 104. Print server 108 also transfers printer data, including printer status and configuration data, from printer 104 to clients 108, 112, 116.

In order to control the flow of data, a local spooler program runs in each client 108, 112, 116. In one embodiment, the spooler program is part of an operating system such as Windows 98 or Windows NT from Microsoft Corporation of Redmond Washington. A user installed port driver in a port monitor interfaces with the print spooler to convert the data from the print spooler into a format suitable for a corresponding printer. For example, when communicating with printers made by Xerox Corporation of Stamford, Connecticut, a Xerox TCP/IP port monitor may be installed on

the server computer to enable formatting of documents to conform to a LPR or Port 9100 protocol. The print monitor maintains data on printer configuration and printer status. Examples of printer configuration data include information on printer set ups such as duplex information, tray and envelope information that may indicate paper sizes and colors, preprinted header information on the paper and the like. Examples of printer status data include information on whether the toner is low or whether a paper jam exists in the printer. For purposes of this application, the term "summary printer data" will be used to include both printer configuration data and printer status data.

Single direction print monitors only allow data flow from the client computer to the printer. Thus updates to printer configuration or status are manually entered by the end user. As previously discussed, such data entry is prone to errors. Bi-directional print monitors enable a printer to automatically update monitor data. However, the Microsoft spooler architecture in Windows NT and Windows 98 was not designed to support bi-directional print monitors. Thus, in current embodiments of bi-directional print monitors operating in a Microsoft operating systems, the print device driver obtains the IP address of the printer and directly addresses the printer circumventing the operating system print spooler.

Circumventing the operating system print spooler breaks operating system protocols for the current generation of Microsoft Windows operating systems. Furthermore, circumventing the operating system print spooler creates additional problems in Windows NT networks that utilize print servers to maintain and monitor printer settings. In particular, when a print server is used to maintain printer settings, client computers connected to the print server are unable to change or otherwise update the stored printer configuration settings because typically only a print server administrator is authorized to change print server local configurations.

In order to implement a bi-directional print driver in a Microsoft Windows operating system and allow upgrading of printer configurations in a print server, one embodiment of the current invention utilizes a polling method of updating data. Figure 2 is a flow diagram illustrating the communication of data in a polling "push" architecture.

In block 204, a poll device, which may be implemented in the port monitor, periodically transmits a request signal to the printer. A server administrator may set the time interval between periodic request signals. Typical printer data requests may include, but are not limited to, configuration and printer status data. Examples of printer configuration data include paper types, paper size, paper finishes (glossy or matte), and envelope trays. Printer status data typically describe a state of printer operation. Examples of printer status data include whether a printer is ready, that the printer is jammed, or that toner levels are low. In block 208, the printer transmits a response signal that answers the request signal. The response signal provides the requested data.

In block 212, a processor compares the printer data received in response to the poll request with stored data from previous poll requests. If the polled data matches the stored data, no change has occurred, no control flags are set, and the system waits a preset time interval in block 214 before transmitting another poll request. However, if the received data does not match the stored data, then a change in either printer status or printer configuration has occurred, and corresponding control flags are set in block 216. In one embodiment, a port monitor manager uses the control flag settings and the stored settings to direct future printer instructions. In an alternate embodiment, the port monitor manager may wake up the print driver to effect the configuration changes as shown in block 220.

When the polling device is implemented in a print server, the change in printer configuration or status may be "pushed" through the system. As used herein, the term "push" refers to transferring data back through the system from the printer to a print server, and from the print server to the client. In block 224, the print sever transmits printer configuration and printer status data changes to the client computers that originate printer instructions. One method of communicating such changes is by transmitting control flags.

In Windows NT operating systems from Microsoft, the preferred method of informing the print driver of configuration changes is by using the Print Driver Event API which is defined and described in the Windows NT DDK (Device Driver Kit) published

by Microsoft of Redmond Washington and which is hereby incorporated by reference. In operating systems in which the server registry is not assumed to be the local registry clients request that a Pipe Server thread be used to transfer the control flags or updated printer data from the server registry to the client computers. Examples of operating systems in which the server registry is not assumed to be the local registry include Windows 95 and Windows 98. Thus, in the previously described embodiment, client computers connected to the print server or a print queue automatically receive the updated settings.

Figure 3 illustrates possible data flow paths from a Microsoft Windows NT client system 304 to a plurality of printers 306, 308, 310, 312. A detailed description of the Windows NT print server architecture is provided in the Windows NT DDK which was previously incorporated by reference. An application, 314, typically software that generates text or graphics such as Microsoft Power Point or Microsoft Word creates a document. Application 314 creates a print job by calling a Graphics Driver Interface 316 (GDI) as illustrated by data path 318. Graphics Driver interface 316 creates a spool file. In specialized applications, the application program may directly create spool file without using the GDI as illustrated along data path 320.

Most functions defined by print providers require a printer handle as input. In an example Windows NT operating environment, the client spooler obtains a printer handle by calling a command OpenPrinter in the client Winspool.driv 322. The calling of the OpenPrinter command causes Winspool.driv 322 to call API server 324 (spoolsv.exe). The capabilities of the spooler are defined by the API functions available.

A client spooler router 326 (spoolss.dll in Windows NT) calls each print provider's OpenPrinter function until one of the print providers supplies a printer handle and a return value indicating the print provider recognizes the specified printer name. The printers called may be either local printers, remote printers running Windows NT and remote printers running other operating systems. Local printers are called via a local connection using localspl.dll 328. Windows NT print servers may be called via a remote connection using Win32spl.dll 330. Print servers that utilize non Windows compatible

operating systems may be called using other Provider DLLs 332. Examples of Non windows DLLs supported by a Windows NT and/or Windows 2000 client system include, but are not limited to, nwprovau.dll for Novell NetWare print servers and inetpp.dll for HTTP print providers that handle print jobs sent to a URL.

A Kernel-mode or Port Driver stack 334 directs the open printer call from the client system 304 to local printer 306, to remote printer 307, to networked servers 336, 338 coupled to corresponding printers 308, 310, and to Windows NT server system 340 coupled to corresponding printer 312. Windows NT server system 340 receives the remote procedure call interface (RPC) signal 342 from the client system 304 at the remote print server router 344. The printer server router 344 generates its own OpenPrinter function 346 that propagates through server Localspl.dll 348 and the print server Kernel-mode Port Driver Stack 350 to printer 312.

The printer spooler router 344 calls each printer's 312 OpenPrinter function until one of them supplies a printer handle and a return value indicating the print 312 recognizes the specified printer name. The printer spooler router 344 then returns its own handle to the client system 304. The print router handle includes both the printer handle and the print server handle. This print router handle is returned to the application 314. The transferred handles allows application 314 to direct subsequent calls to the correct print server and printer.

Figure 4 shows a port monitor architecture 400 as implemented in one embodiment of the invention. In one embodiment of the invention, a server, using the port monitor of Figure 4, transmits inquiries regarding printer status and configuration to a printer using a polling method. When a change in printer status or configuration occurs, the port monitor updates stored printer configuration and status data in a registry of the server. Some client operating systems, such as Windows NT utilizes the server registry as the local registry. However, other client operating systems, such as Windows 9x series operating systems including Windows 95 and Windows 98 maintain a separate local registry. In such systems, the client may periodically poll the server to determine printer configuration and/or status changes. When such changes are detected, the client

may retrieve the printer configuration or status changes from the server via a server thread.

Block 402 includes protocol support 404 hardware and software that receives streams of print data from each port cell such as a first port 406 and a second port 408. Protocol support 404 converts the data into a print data stream that matches an acceptable printer protocol for output to a printer. Typical IEEE accepted protocols include LPR (Line print protocol), LPR with byte counting and AP Socket 9100 although other protocols may also be possible.

Because the print system is a dynamic system, printers and clients may be added to the network. Block 410 of Figure 4 includes components used to add ports to the print server. In block 410, Port Wizard software 412 communicates with an auto printer discovery dynamic link library (DLL) 414. Discovery DLL 414 utilizes discovery APIs 416 to determine attached or available printers. After determining available printers, Port Wizard software 412 adds ports using add port subroutine 418. Configure port subroutine 420 properly configures the added ports. When the print spooler 424 is part of an operating system, both the port management software of block 402 and the add port software of block 410 use port monitor application program interfaces (APIs) shown in block 422 to format and transmit requests to the print spooler. When the operating system is a Windows operating system from Microsoft Corporation of Redmond, WA, the port monitor APIs in block 422 are Microsoft defined APIs used to communicate with the Windows Print Spooler.

In Figure 4, Port Device Data Manager 426 interfaces with a port monitor printer manager 428. In one embodiment of the invention, both port device data manager 426 and port monitor printer manager 428 are part of a print server that transfers information between a printer and client computers.

Port device data manager 426 includes a shared locked memory 430 that maintains configuration data, status data and control data for each printer coupled to the print server. A port manager API 440 manages communications with a printer using a standard communications protocol such as SNMP. SNMP communication is controlled

stored in printer list 458 and available port data stored in port list 460. Printers identified in list 458 typically have a corresponding entry in registry 468 corresponding to the printer. Each printer entry in printer list 458 may include, but is not limited to, printer names, device ids, port names, port handles, configuration Ids, status Ids, and printer events. Port list 460 may include, but is not limited to data pertaining to each port such as port handles and previous control states.

Print manager thread 454 summarizes data received from printer list 458, port list 460 and shared locked memory 430 and forwards the summarized data to a summarized data 469 area in printer registry 468. In addition, print manager thread 454 alerts printer driver 462 of the changes. The print driver converts the summarized data 469 to an internal format 471 compatible with the print driver. The internal format data 471 is stored in a corresponding area of printer registry 468, The operation of a typical print driver is described in the Microsoft Windows NT DDK which has been incorporated by reference.

When client 464 is a client that maintains a local print driver and thus does not automatically assume use of a server print driver, (examples of such operating systems include Windows 9x clients referring to clients computers that operate using Microsoft Windows 95 and/or Windows 98) the client 464 requests that pipe server thread 466 retrieve summarized data 469. Pipe server thread 466 may perform or facilitate both client transmission of data request commands as well as pipe server thread 466 generated responses of configuration and status information.

Some clients may utilize operating systems that assume the use of a print server driver as the local driver. One example of such an operating system is a Windows NT operating system. Each Windows NT client, such as client 470, may directly access data from registry 468.

Figure 5 shows one example of a user interface that may be implemented for controlling bidirectional communications. In screen 504, a summary 506 of port setting is provided. Any of these values may be modified by selecting back button 508 to

change a selected characteristic. Completion of the port configuration may be achieved by hitting finish button 512.

Additional details of configuring the port are illustrated in Screen 516 of Fig 6. Bi-directional settings section 520 of screen 516 allow users to enable or disable bi-directional communications. Screen 516 also allows user defined timer intervals between printer configuration updates and printer status updates by filling in the corresponding time interval boxes 524, 528.

It will be appreciated that the foregoing description is intended to be illustrative. Variations and modifications of the descriptions provided herein will present themselves to those skilled in the art. For example, the focus of the descriptions has been on the transfer of data between printers and clients running specific Windows operating systems. However, other operating systems which use architecture similar to the described operating systems may also implement the described invention. As a further example, different types of printer data besides the examples of printer status and printer configuration data described may be transferred. Accordingly, the present description should not be read as limiting the scope of the claims except and unless indicated to the contrary.